UNIVERSITETET I OSLO

Det matematisk-naturvitenskapelige fakultet

«Midterm» i:	**INF 4130: *Algoritmer: Design og effektivitet***
Eksamensdag:	**1. november 2011**
Tid for «midterm»:	**Kl. 09:00 – 13:00 (4 timer) [124%, 100%=3]**
Oppgavesettet er på:	**6 sider**
Vedlegg:	**Ingen**
Tillatte hjelpemidler:	**Alle trykte og skrevne**

Kontrollér at oppgavesettet er komplett før du begynner å besvare spørsmålene.

Les oppgavene nøye, og lykke til!

Oppgave 1 Dynamisk programmering (15 %)

Vi skal se på følgende problem: Gitt to bokstavsekvenser *S* og *T*. Vi skal avgjøre om man kan lage *S* ut fra *T* ved kun å fjerne tegn fra *T*. Om *S* = `abc` og *T* = `aacbbac` er dette mulig, men ikke om T = `aacbba`.

Spørsmål 1.a (7%)
Angi hvordan man kan løse dette problemet med dynamisk programmering. Angi her hva slags tabell du vil bruke, hvordan den generelle reglen for utfylling er, og hvordan du vil initialisere tabellen. Skisser også et program som gjør dette.

Spørsmål 1.b (8%) (Vente med denne til slutt?)
Forklar hvordan du kan forandre din algoritme fra 5.a slik at den løser følgende problem: Man kan fremdeles stryke tegn i *T*, men får ikke lov til å stryke to tegn som står etter hverandre i (den opprinelige) *T*. Er det da mulig å få fram *S* fra *T*? (Hint: Forsøk å bruke to varianter av «TRUE» i tabellen.)

Oppgave 2 Dynamisk programmering (13%)

En lang tømmerstokk av lengde l skal kappes i mindre deler på et sagbruk. På grunn av kvist, stokkens tykkelse, hva delene skal brukes til ol., er posisjonene hvor stokken skal kappes bestemt på forhånd. Disse er: p_1, p_2, ..., p_{n-1}, målt fra den venstre enden av stokken. I tillegg lar vi p_0=0 være den venstre enden av stokken, og p_n=l den høyre. Etter å ha delt den opprinnelige stokken én gang har vi to nye stokker, etter å ha delt den to ganger har vi tre, osv.

På sagbruket går stokkene på transportbånd, slik at det tar tid t å kappe en stokk som er t lang uansett hvor den kappes. Stokken kan bare kappes én gang per gjennomkjøring, de to nye delene må eventuelt prosesseres på nytt, hver for seg, om de skal kappes ytterligere. Stokker som ikke skal kappes mer kjøres ikke gjennom saga, disse tar altså tid 0.

Eksempel: En 10 meter lang stokk skal kappes ved p_1=2 meter, p_2=5 meter, p_3=8 meter. Kapper vi først ved p_3, så ved p_2, og så ved p_1, vil dette ta tid 10+8+5=23. Kapper vi først ved p_2, og deretter p_1 og p_3, vil det ta tid 10+5+5=20, altså noe raskere.

Oppgaven er: Definér en formel (en rekurrensrelasjon) som kan brukes i dynamisk programmering for å finne raskeste måte å kappe den opprinnelige stokken på, og vis hva initialbetingelsene vil være. Skissér gjerne et program, om du syns det er enklere enn å definere en rent matematisk formel.

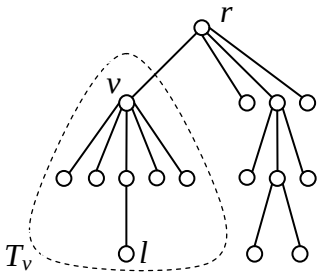
Oppgave 3 Dynamisk programmering (*21 %*)

En *uavhengig mengde* av noder i en graf er en delmengde av noder det ikke går noen kanter mellom. To nabonoder i grafen kan altså ikke begge være med i en slik uavhengig mengde.

Figur 1. De svarte nodene i treet til venstre utgjør en uavhengig mengde – det går ingen kanter mellom noe par av svarte noder. Dette er faktisk også en største uavhengig mengde – det finnes ingen uavhengig mengde større enn elleve noder i det venstre treet. De svarte nodene i treet til høyre utgjør ingen uavhengig mengde – det går kanter mellom to og to av dem.

Å finne den største uavhengige mengde av noder i en generell graf er et NP-hardt problem, i denne oppgaven skal vi derfor kun se på trær. I trær kan vi nemlig bruke dynamisk programmering til å finne størrelsen av største uavhengige mengde (og også den faktiske mengden, om vi ønsker). [Sidebemerkning: Å *kun* se på trær er ikke en *så* stor forenkling som det kanskje kan høres ut. Det finnes nemlig en mye større klasse av tre-aktige grafer hvor den samme metoden kan brukes. Vanlige trær er spesialtilfeller av disse grafene.]

La T være treet vårt og r være roten. For hver node v i treet, la T_v være sub-treet med v som rot (se Figur 2). Husk at det for enhver node v vil være to muligheter når vi forsøker å danne en uavhengig mengde: enten å ta v med i mengden eller å la det være.



Figur 2. Et tre T med rot r, en løvnode l, en intern node v, og subtreet T_v med rot i v.

Spørsmål 3.a (7 %)
Om vi skal beregne størrelsen av største uavhengige mengde av noder i *T*, *bottom-up*, må vi opplagt starte med løvnodene. Vi tenker oss at de beregnede verdiene lagres i nodene i treet, slik at vi slipper å bruke en tabell. For en løvnode *l*, og sub-treet T_l, skal vi beregne to verdier:

l.m størrelsen av den største uavhengige mengde i T_l som inneholder noden *l*,
l.u størrelsen av den største uavhengige mengde i T_l som *ikke* inneholder noden *l*.

Hva blir disse verdiene for en løvnode *l*? (Merk at vi *kun* ser på sub-treet T_l.)

Spørsmål 3.b (7 %)
Når løvnodene er ferdige, kan disses foreldre gjøre sine beregninger, og så videre oppover i treet. Generelt må en node vente til alle dens barn er ferdig før den kan gjøre sine egne beregninger. De resterende nodene *v*, med korresponderende sub-trær T_v, skal også beregne:

v.m størrelsen av den største uavhengige mengde i T_v som inneholder noden *v*,
v.u størrelsen av den største uavhengige mengde i T_v som *ikke* inneholder noden *v*.

Angi med enkel pseudokode hvordan *v.m* og *v.u* beregnes for en node *v* som ikke er et løv. Du skal ikke skrive kode for traversering av treet eller liknende, kun vise hvordan *v* gjør sin beregning basert på verdier som nå allerede er beregnet. (Merk at vi fortsatt *kun* ser på sub-treet T_v.) Du kan anta at *v*s barn ligger i en passende datastruktur, og gå igjennom disse med en for-løkke `FOR i IN v.children DO{}` som aksesserer allerede beregnede verdier for hvert barn *i* av *v*.

Spørsmål 3.c (7 %)
Når løvnodene har beregnet initialverdiene, og de resterende nodene har beregnet sine verdier nedenfra og opp, er vi i prinsippet ferdig. Hvor/hvordan finner vi nå størrelsen av største uavhengige mengde i *T*?

Oppgave 4 Strengsøk (*18 %*)

Vi skal bruke Karp-Rabin-algortimen for å søke etter patternet `666` i strengen `12345666`.

Vi antar at algoritmen arbeider med de vanlige tallene i titallssystemet, som beskrevet i kursboken (Berman & Paul, *Algorithms: Sequential, Parallel, and Distributed*), slik at vi slipper omregning via et annet og mer uvant tallsystem, videre antar vi at algoritmen arbeider modulo 3.

Spørsmål 4.a (12 %)
Vis hvordan algoritmen arbeider stegvis under søket, hva den aktuelle delen av strengen (vinduet) omegnes til modulo 3, og angi hvor det ikke blir match, hvor det blir uekte (spuriøs) match og hvor det blir ekte match.

Du kan gjerne gjøre dette i kort tabell-form, f.eks slik:

1	2	3	4	5	6	6	6	Omregning av vinduet modulo 3	Match?
1	2	3						123 mod 3 = 0	?
	2	3	4					?	?
⋱								⋮	⋮

Spørsmål 4.b (6 %)
Hvordan forholder det antall uekte (spuriøse) matcher du fant i 2.a seg til det forventede antall uekte (spuriøse) matcher i uniformt fordelte strenger når man arbeider modulo 3?

Oppgave 5 A-søk og heuristikk-funksjoner (16%)*

Spørsmål 5.a (8%)
Vi er gitt et enkelt brikkespill bestående av n hvite (**H**) og n sorte (**S**) brikker, samt en åpen rute, som vist på figuren under.

1	2	…	n	$n+1$	$n+2$	$n+3$	…	$2n+1$
S	**S**	**…**	**S**		**H**	**H**	**…**	**H**

Målet er å flytte alle de hvite brikkene over på venstre side av brettet og de sorte på høyre side. Hvor den åpne ruten havner er uinteressant, og vi bryr oss heller ikke om den innbyrdes rekkefølgen av de hvite og de sorte brikkene. I slutt-tilstanden skal det altså ikke være noen sorte brikker til venstre for (med lavere indeks enn) noen hvit brikke. Vi kan gjøre to slags flytt:

- Flytte en nabo-brikke av den åpne ruten inn i den åpne ruten. Dette har kostnad 1.
- Hoppe over nøyaktig én brikke, inn i den åpne ruten. Dette har kostnad 2.

Vurder om følgende forslag til heuristikker, angitt for en generell spill-situasjon, er monotone eller ikke:

1. Antall enkelt-flytt (av kostnad 1) som trengs for å flytte alle H-brikkene helt til venstre (om det ikke var noen S-brikker) pluss tilsvarende antall enkeltflytt det tar å flytte alle S brikker helt til høyre.
2. For hver S-brikke teller vi opp antall H-brikker til høyre for denne. Så summerer vi opp disse verdiene for alle S-brikkene.

Spørsmål 5.b (8%)
Når vi arbeider med A*-søk og heuristikk-funksjoner, hender det ofte at vi kan bruke forenklede versjoner av vårt opprinnelige problem til å lage heuristikk-funksjonene våre. Professor Max har på denne måten laget tre ulike heuristikk-funksjoner for et problem han arbeider med: h_1, h_2 og h_3. Han har verifisert at alle de tre funksjonene er monotone, men han har ikke klart å avgjøre hvilken han skal bruke. For noen noder ligger heuristikk h_1 nærmest den faktiske kostnaden, for andre h_2 eller h_3. Professorens datter foreslår at han benytter funksjonen

$$h(x) = \max\{h_1(x), h_2(x), h_3(x)\} .$$

Kan han det? Forklar hvorfor/hvorfor ikke.

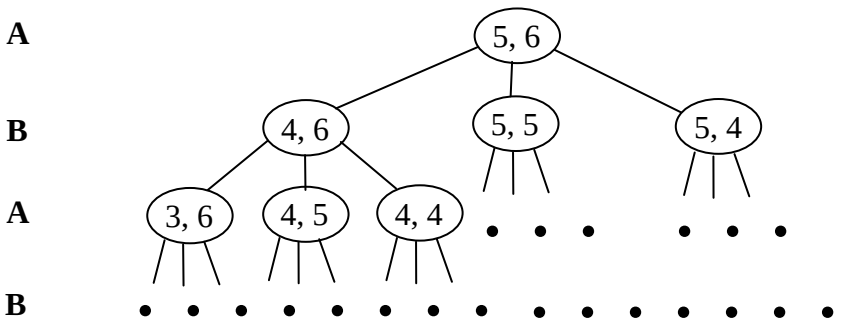
Oppgave 6 Spilltrær (20 %)

Et spill for to personer er som følger: Det er to bunker med fyrstikker *F* og *G*, med h.h.v. *f* og *g* fyrstikker. Vi skriver en gitt situasjon slik: (*f, g*), eller med hel ring rundt. Det er fra hver situasjon maksimalt tre mulige trekk:

- Fjerne en fyrstikk fra *F*,
- Fjerne en fyrstikk fra *G*,
- Fjerne to fyrstikker fra *G*.

Et trekk er bare lovlig om det nok fyrstikker i den aktuelle bunken. Spillerne er **A** og **B**, de trekker annenhver gang, **A** trekker først. Den spilleren som er i trekket når situasjonen er (0, 0) har vunnet (altså, den som tar siste fyrstikken taper).

Vi skal tegne opp spilltrær for dette spillet. Subnodene til en node skal alltid tegnes fra venstre mot høyre i den rekkefølge de tre eller færre mulige trekk er angitt over. Et tre fra startsituasjonen (5, 6) kan for eksempel begynne fra toppen slik:



Figur 3. Toppen av spilltreet med (5, 6) som startsituasjon.

Spørsmål 6.a (4 %)
Tegn opp hele spilltreet fra startsituasjonen (1, 3). Ikke gjør noe forsøk på å slå sammen like noder.

Spørsmål 6.b (4 %)
Vi vil merke vinstnoder for **A** (altså, der **A** har en vinnende strategi) med **+** og vinstnoder for **B** med **–**. Merk ut fra dette alle nodene i spilltreet fra oppgave 1.a med enten **+** eller **–**.

Spørsmål 6.c (4 %)
Finnes det en vinnende strategi for den som skal trekke fra situasjon (1, 3)?

Spørsmål 6.d (4 %)
Finnes det en vinnende strategi for den som skal trekket fra situasjon (1, 2)? Forsøk å bruke informasjon du allerede har samlet.

Spørsmål 6.e (Uavhengig av 1.a – 1.d) (4 %)
Vi vil bruke alfa-beta-avskjæring ved gjennomgang av et spilltre, men vi er så uheldige at vi alltid fra en situasjon først ser på et dårligst mulig trekk for den som skal trekke. Kari påstår da at det umulig kan bli noen alfa-beta-avskjæring i det hele tatt? Ola mener at dette er feil. Hvem har rett? Forklar!

Oppgave 7 Diverse smågreier (21%)

Spørsmål 7.a (7%)
Vi har følgende to venstrevidde heaper (leftist heaps) L1 og L2:

Vis resultatet av å spleise L1 og L2 (`merge(L1,L2)`).

Spørsmål 7.b (7%)
Tegn et suffix-tre for strengen `GAACAACT`.

Spørsmål 7.c (7%)
Beregn shift-verdier for strengen `GAACAACT` slik det gjøres i den forenklede Boyer-Moore-algoritmen (Horspool-algoritmen). Ta utgangspunkt i at alfabetet består av de vanlige norske, store bokstavene (A—Å).